USENIX Security '25 Artifact Appendix: *Found in Translation*: A Generative Language Modeling Approach to Memory Access Pattern Attacks

Grace Jia
Yale University

Alex Wong
Yale University

Anurag Khandelwal
Yale University

A Artifact Appendix

A.1 Abstract

Confidential computing environments (CCEs) offer a convenient way for privacy-sensitive applications to ensure confidentiality and integrity for data and computations offloaded to the cloud. However, the Operating System (OS) stack is still controlled by the cloud provider and manages key system services such as memory paging. Several recent works have demonstrated that these services can leverage the side channel of page access patterns to reconstruct the application's private data. However, prior attacks are limited to identifying accesses to a specific application-level object by a unique access or sequence of accesses to OS-level pages. Moreover, they tend to ignore correlations in access patterns — a common occurrence in most real-world applications — leaving untapped critical side channel information for improving attack accuracy.

We propose a novel attack approach, named *Found in Translation* (FɪT), that leverages access correlations across pages in cloud applications using a generative language model. Our key insight is that there are strong parallels between application page access patterns and grammatical structures in natural languages, making language modeling an excellent fit for reconstructing sensitive application data with high accuracy. Our major results are the high accuracy of the attack compared to prior state-of-the-art approaches in predicting object-level access sequences on three privacy-sensitive applications: DLRM inference for recommendations, LLM inference for medical diagnosis, and HNSW index lookup for semantic search. We also include results on our attack's robustness against errors in the recorded page access sequences and the latency that OS-level tracking and recording page accesses add to the execution times of the victim applications.

A.2 Description & Requirements

This artifact contains the following: (i) the source code to train and evaluate the language model used in FɪT, along with code to run the compared baselines — the IHOP attack and the Naive Bayes classifier; (ii) the preprocessed datasets used in our evaluation to train and test all compared attacks, FɪT's trained model weights for each evaluated application, and our measurements of application execution times with and without page access tracking; (iii) the scripts required to reproduce our experiments and a comprehensive README with instructions to set up and run the FɪT attack pipeline.

A.2.1 Security, privacy, and ethical concerns

This artifact does not take destructive steps, nor does it disable security mechanisms. It does not pose any privacy or ethical concerns to evaluators.

A.2.2 How to access

Our artifact is available on Zenodo and can be accessed at https://doi.org/10.5281/zenodo.15602651.

A.2.3 Hardware dependencies

We evaluated our FɪT attack using a single NVIDIA GeForce RTX 4090 GPU; however, we expect FɪT to run on any NVIDIA GPU with at least 16 GB of memory. We also provide an option for CPU-only platforms to run our experiments on a smaller subset of test samples.

The compared attacks, IHOP and Naive Bayes, require only CPU hardware to evaluate. We have previously run their experiments on AMD EPYC 7302P 16-core processor, as well as Apple M2 Pro 12-core processor with 16 GB RAM.

A.2.4 Software dependencies

We conducted experiments involving FɪT on Ubuntu 24.04.1 LTS. Experiments involving compared attacks have been conducted on Ubuntu 22.04.2 LTS and macOS Sonoma 14.6.1. All attacks require Python3, and the plotting script requires LaTeX to be installed. We use Conda for environment management and document all required packages in the file 'requirements.txt' in the root directory of the artifact.

A.2.5 Benchmarks

We evaluate our compared attacks on page access traces collected from three applications: a Deep Learning Recommendation Model (DLRM), a Large Language Model (LLM), and a Hierarchical Navigable Small World index (HNSW); more details about the datasets can be found in our paper. All required data and model weights are provided on Zenodo.

A.3 Set-up

We assume that the system has at least 22 GB of available disk: 17 GB for the artifact, and 5 GB for the Conda environment.

A.3.1 Installation

We recommend using Conda to set up our experimental environment. If Conda is not installed on the system, you can quickly set up Miniconda following these instructions. You may need `wget` or `curl` to download the installer.

First, create a new Conda environment with Python 3.12:

```
conda create -n fit -y python=3.12
conda activate fit
```

Next, install the required dependencies:

```
pip install -r requirements.txt
```

A.3.2 Basic Test

The following script runs FIT, IHOP, and the Naive Bayes baseline on a single test sample from our LLM dataset:

```
bash scripts/run_basic_test.sh
```

We expect the tests to take around 5 minutes in total. Each test populates the ‘data/llm/eval’ directory with a results file — respectively, ‘llm_nitro.csv’, ‘nb_llm.csv’, and ‘ihop_llm.pkl’. Before running our experiments, please remove these files with `rm data/llm/eval/*`.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): FIT achieves significantly lower hamming distances (i.e., more accurate predictions) than the compared Naive Bayes and IHOP baselines across all three use cases, DLRM, LLM, and HNSW. This is demonstrated by experiments (E1), (E2), and optionally (E3), the results of which are shown in Figure 7.

(C2): Attack efficacy improves for all three attacks given a one-to-one mapping of OS-level pages and application-level objects, but FIT is still more accurate than IHOP. This is also demonstrated by experiments (E1), (E2), and (E3), and the relevant results are shown in Figure 8.

(C3): FIT is robust against measurement errors in the input page access traces. This is demonstrated by experiment (E4), the results of which are shown in Figure 9.

A.4.2 Experiments

Preparation: Before running each script, set the current directory to the root directory of the artifact and activate the `fit` Conda environment.

Results: If successful, each experiment will save its results in directories corresponding to each use case: ‘data/dlrm/eval/’, ‘data/llm/eval/’, and ‘data/hnsw/eval/’.

(E1): *[*FIT *Attack Efficacy] [3.5 GPU compute-hours]:* Runs the inference phase of FIT on test samples from the three use cases for (C1) and the one-to-one use case for (C2).

Execution: To run the experiment with the paper’s test dataset sizes for DLRM, LLM, and HNSW:

```
bash scripts/run_fit.sh \
100000 50000 2600
```

If only CPU is available, we recommend running the DLRM and LLM experiments with fewer test samples:

```
bash scripts/run_fit.sh \
1000 5000 2600 --use-cpu
```

Results: The results of the FIT experiment are saved as the following files in the result directories corresponding to their use case: ‘dlrm_nitro.csv’, ‘dlrm_sgx.csv’, ‘llm_nitro.csv’, ‘llm_sgx.csv’, ‘hnsw_nitro.csv’, ‘hnsw_sgx.csv’ for (C1), and finally ‘dlrm_1_1.csv’ for (C2). Each row of the CSV contains an access sequence predicted by the FIT model and the corresponding ground-truth target sequence. After collecting the results of (E2), the following script can reproduce Figures 7 and 8 from our paper and save them to the ‘plots’ directory:

```
python3 scripts/plot.py --fig 7
python3 scripts/plot.py --fig 8
```

(E2): *[Naive Bayes Attack Efficacy] [1.5 CPU compute-hours]:* Runs the Naive Bayes baseline on the train and test data of the three use cases for (C1) and the one-to-one use case for (C2).

Execution: `bash scripts/run_nb.sh`

Results: Like (E1), results of the Naive Bayes runs are saved in the following CSV files: ‘nb_dlrm.csv’, ‘nb_llm.csv’, ‘nb_hnsw.csv’, and ‘nb_dlrm_1_1.csv’. Each row of each CSV is a tuple of the predicted and target sequences.

(E3): *[IHOP Attack Efficacy] [56 CPU compute-hours]:* Runs the IHOP attack on the train and test data of the three use cases (C1) and the one-to-one use case (C2). The compute-hours are estimated based on prior runs with the AMD setup, but we have observed an up to 2x speedup on the Apple M2 Pro.

Execution: `bash scripts/run_ihop.sh`

Results: Each run of IHOP outputs a list of predicted and target sequences, which are saved as the following pickle files in the result directories: 'ihop_dlrm.pkl', 'ihop_llm.pkl', 'ihop_hnsw.pkl', and 'ihop_dlrm_1_1.pkl'. As this experiment is relatively time-intensive, our artifact provides results from our runs of IHOP in the parent directories 'data/dlrm/', 'data/llm/', and 'data/hnsw/'. These provided results are automatically used by the plotting script if there is insufficient time to run (E3).

(E4): *[*FiT *Practical Considerations] [6.5 GPU compute-hours]:* Runs the inference phase of FiT on test samples from the three use cases with varying measurement error rates: $1\%, 3\%, 5\%, 7\%$, and 10% (C3).

Execution: To run the experiment:

```
bash scripts/run_fit_sensitivity.sh
```

Like (E1), CPU-only options are available:

```
bash scripts/run_fit_sensitivity.sh \
1000 5000 2600 --use-cpu
```

Results: The results of each run of FiT inference are saved as a CSV, with each row containing the access sequence predicted by the FiT model and the corresponding ground-truth target sequence. The following script can then recreate Figures 9 and 10:

```
python3 scripts/plot.py --fig 9
python3 scripts/plot.py --fig 10
```

A.5 Version

Based on the LaTeX template for Artifact Evaluation V20231005. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/usenixsec2025/.